Spark 框架结合分布式 KNN 分类器的网络大数据分类处理方法 *

曹 瑜[1]，王 楠[2,3]，徐志超[2]

(1. 哈尔滨金融学院 计算机系，哈尔滨 150030; 2. 吉林财经大学 管信学院，长春 130117; 3 吉林大学 计算机学院，长春 130012)

摘 要：针对现有大数据分类方法难以满足大数据应用中时间和储存空间的限制，提出了一种基于 Apache Spark 框架的大数据并行多标签 K 最近邻分类器设计方法。为了通过使用其他内存操作来减轻现有 MapReduce 方案的成本消耗，首先，结合 Apache Spark 框架的并行机制将训练集划分成若干分区；然后在 Map 阶段找到待预测样本每个分区的 K 近邻，进一步在 reduce 阶段根据 map 阶段的结果确定最终的 K 近邻；最后并行地对近邻的标签集合进行聚合，通过最大化后验概率输出待预测样本的目标标签集合。在 PokerHand 等四个大数据分类数据集上进行实验，提出方法取得了较低的汉明损失，证明了其有效性。

关键词：分类处理；Apache Spark；并行机制；数据挖掘；汉明损失；K 最近邻

中图分类号：TP391　　doi: 10.3969/j.issn.1001-3695.2018.05.0296

Network big data classification processing method based on Spark and distributed KNN classifier

Cao Yu[1], Wang Nan[2, 3], Xu Zhichao[2]

(1. *Dept. of Computer, Harbin Finance University, Harbin* 150030, *China*; 2. *Institute of Management & Credit Jilin University of Finance & Economics, Changchun* 130117, *China*; 3. *School of Computing Jilin University, Changchun* 130012, *China*)

Abstract: Aiming at the limitation that the existing big data classification methods can not meet the time and storage space in big data applications, a design method of big data parallel multi-label k-nearest neighbor classifier based on Apache Spark framework is proposed. In order to reduce the cost of the existing MapReduce scheme by using other memory operations, first, the training set is divided into several partitions in conjunction with the parallel mechanism of the Apache Spark framework. Then in the Map stage, the K nearest neighbors of each partition of the sample to be predicted are found, and in the Reduce phase, the final K nearest neighbors are determined according to the results of the Map phase. Finally, the neighboring tag sets are aggregated in parallel, and the target tag set of the sample to be predicted is output by maximizing the posterior probability. Experiments were conducted on PokerHand et al. 's four big data classification datasets. The proposed method achieved a lower Hamming loss and proved its effectiveness.

Key words: classification processing; Apache Spark; parallelism; data mining; hamming loss; K-nearest neighbor

0 引言

在过去几年中，大数据已成为一个焦点词汇，大规模的信息处理已成为一项必要且关键的任务[1~3]。技术的不断革新以及消费方式的改变，带来了商业的信息化发展。而商业的信息化发展导致了可用数据量的严重增加。另一方面，社交媒体，生物医学和娱乐活动每天同样产生大量数据，如果没有适当的知识提取过程，这些数据是无用的。因此迫切需要一种便捷的工具能够利用这些数据。K 近邻算法(K-nearest neighbor algorithm，kNN)[4]是一种直观且有效的非参数模型，常用于分类和回归。

一些研究在 MapReduce 过程中引入了 kNN 分类器，但他们的目的不是要执行精确的 KNN 分类，而是使用部分 KNN（KNN 应用于训练数据的子集）对大数据集进行聚类。文献[5]提出了一种新颖的方法，将 KNN 和主成分分析相结合来对大数据集进行聚类。文献[6]提出一种方法，该方法有两个不同的

收稿日期：2018-05-07；**修回日期：**2018-07-03　**基金项目：**国家自然科学基金资助项目（61702213）；吉林省教育厅"十三五"科学技术研究（JJKH20180463KJ）

作者简介：曹瑜（1979-），女，辽宁海城人，讲师，硕士，主要研究方向为大数据、数据挖掘（caoyu134638@126.com）；王楠（1980-），女，吉林吉林人，副教授，博士，主要研究方向为数据挖掘与机器学期；徐志超（1977-），男，吉林长春人，工程师，硕士，主要研究方向为大数据、人工智能.

阶段,在第一阶段使用 K-means 来分隔整个数据集的不同部分,第二阶段计算每个分组中的 KNN 来提供近似的结果。还有一些研究没有针对分类或回归任务，而是提出在 MapReduce 中执行 KNN 进行查询的分布式计算。例如，文献[7]提出在两阶段 MapReduce 过程中应用 KNN-join（精确或近似）查询;文献[8]针对基于 KNN 的图像分类提出了迭代 Hadoop MapReduce 过程。然而，这种方法为每个单独的测试实例迭代执行 MapReduce，造成了较大的时间消耗。文献[9]提出了一种 Hadoop MapReduce 过程，可以将大量测试样本进行分类，降低了 Hadoop 的启动成本。然而该方法的分类准确性较低，还有待进一步完善。

在本文中，提出了一种基于 Apache Spark 框架的大数据并行多标签 K-最近邻分类器设计方法。在本文的实现中，目标是利用 Spark 提供的灵活性，通过使用其他内存操作来减轻现有 MapReduce 方案的成本消耗。为了管理巨大的测试集，根据内存限制，将可能的测试示例的最大数量用于最小化迭代次数。在每次迭代中，将应用 KNN MapReduce 进程，在 map 阶段找到待预测样本每个分区的 K 近邻，并在 reduce 阶段确定最终的 K 近邻。最后并行地对近邻的标签集合进行聚合。

1　海量数据集的分布式处理

1.1　MapReduce 框架

MapReduce(MR)框架是为数据的并行计算提供高度可扩展和灵活的框架，主要是是为了应对日益增多的数字数据[10~12]。在过去的几年里，数据的生成和存储能力已经增长到 PB 级，这是硬件和新型处理技术更成熟的结果。MR 的主要潜力是它提出的计算抽象，其中整个处理被划分成更小的任务类型 Map 和 Reduce，沿着集群均匀分布和处理。从业者只需负责提供这两个功能，避免将处理适配到群集的底层架构或数据的性质。该框架为并行数据处理提供了一个高度可扩展的容错环境。

在 MR 工作流程中管理的数据由⟨ $key,value$ ⟩形式的键-值对表示。每个映射任务的初始输入通常作为原始数据的输入分区，与任意键相关联。每个 Map 任务通过对其应用一个定义的函数来处理其大块数据，这产生了许多新的中间 key-value 输出对。系统通过相应的键收集、组合和排序这些中间输出对，并发送与输入数据共享相同键的匹配对。该输入由一个由公用 key 和包含关联值的列表组成。然后，执行 Reduce 步骤；结果会产生最终输出，通常由新格式化的键、值对组成。图 1 显示了 MR 程序的详细流程。

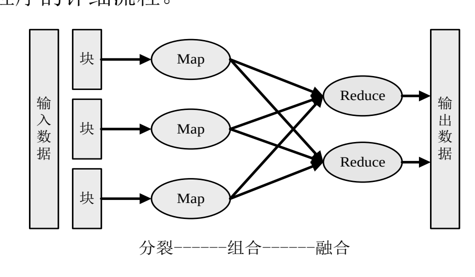


图 1　MR 流的图形表示

这个框架的普及是由 Apache Hadoop 的实现引起的；开放软件社区与许多私营公司共同努力为大数据预处理提供广泛可用的软件堆栈。Hadoop 生态系统非常广泛，但主要是两个主要组件：通用 MapReduce 和 Hadoop 分布式文件系统(HDFS)。它们允许部署廉价的大型计算机集群作为强大的执行引擎以及可靠和容错的分布式数据存储。

1.2　Spark 框架

Spark 框架可以被看做是下一代分布式计算框架，并且是 MapReduce 的扩展[13]。它们之间的关键区别在于 Spark 为正在处理的数据定义了一个新的抽象，称为弹性分布式数据集(RDD)；这使得从业者可以定义额外的数据操作，而不必严格遵守 Map 和 Reduce 功能。但是，MapReduce 范例仍然是 Spark 平台的核心，因为大部分计算都遵循相同的模式，即将几个函数单独应用于每个数据单元，然后通过网络混合进行组合。

Spark 成为分布式数据处理实际标准的主要动机依赖于体系结构的差异：虽然 Hadoop MR 依靠硬盘驱动器来获取每个操作之间的中间数据，但 Spark 专注于更快的主内存来维护其数据结构。这允许定义更复杂的执行路径以及增加对迭代过程的支持。图 2 展示了 Spark 算子和数据空间。

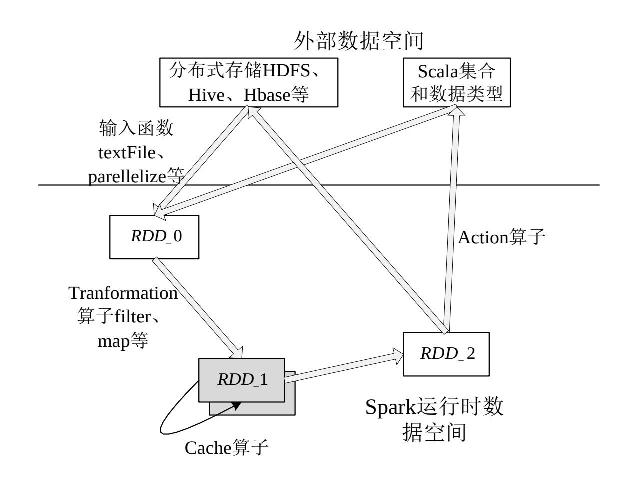


图 2　Spark 算子和数据空间

最近基于云计算的技术为人们提供了处理此问题的理想环境。MapReduce 框架及其在 Hadoop 中的开源实现是基于数据局部性原理处理数据密集型应用的工具，并且已在数据挖掘中广泛传播。但是，研究人员发现 Hadoop Mapreduce 在设计可伸缩机器学习工具方面存在一些限制。对于跨多个步骤共享数据的应用程序（包括迭代算法或交互式查询），MapReduce 效率不高。最近出现了多个用于大规模数据处理的平台来克服 Hadoop MapReduce 的问题。其中 Spark 已成为大数据中执行更快分布式计算的最灵活和最强大的引擎之一。该平台允许用户程序将数据加载到内存中并反复查询，使其更适合在线，迭代或数据流算法。

2　KNN 分类器

kNN 算法是一种非参数方法，可用于分类和回归任务[15]。kNN 算法的表示如下：

设 Ds 为训练数据集，Ts 为测试集，它们分别由数量为 n 和 t 的样本组成。每个样本 x_p 都是一个多元组 $(x_{p1},x_{p2},...,x_{pD},\omega)$，其中，$x_{pf}$ 表示第 p 个样本的第 f 个特征的值。样本属于类别 ω，是由 x_p^{ω} 和一个 D 维空间确定。对于 Ds 集合，类别 ω 是已知的，而这一点对于 Ts 集合是未知的。

对于包含在 Ts 集合中的每个样本 x_{test} 测试，kNN 算法搜索 Ds 集合中 k 个最接近的样本。因此，KNN 计算 Ds 中所有样本 x_{test} 之间的距离。欧几里得距离是用于此目的的最广泛使用的度量。根据计算出的距离，训练样本按升序排列，取 k 个最近样本 $(neigh_1,neigh_2,...,neigh_k)$ 然后，它们被用来计算最主要的类别标签。k 的选择值可能会影响该技术的性能和噪声容限。尽管 KNN 在各种各样的问题中表现出色，但缺乏管理大型 TR 数据集的可扩展性。处理大规模数据的主要问题是：

a)运行时间。查找单个测试实例的最近邻居训练样本的复杂度为 $O((n{\cdot}D))$，其中 n 是训练实例的数量，D 是特征的数量。由于它需要对计算距离进行排序，所以在找到 k 个邻居时，这在计算上更复杂，因此需要额外的复杂度 $O(n{\cdot}log(n))$。最后，这个过程需要重复每个测试例子。

b)内存消耗。为了快速计算距离，KNN 模型要求将训练数据存储在内存中。当 Ds 和 Ts 集太大时，它们可能很容易超过可用的 RAM 存储器。

这些缺点促使将 KNN 的处理分布在节点集群上。在文献中，可以找到一系列在 MapReduce 上执行 KNN 的方法。虽然 KNN 分类器旨在提供预测的类，但 KNN 连接本身会输出邻居进行单个测试。因此，这些方法不能用于分类。图 3 展示了分布式 KNN 的流程。

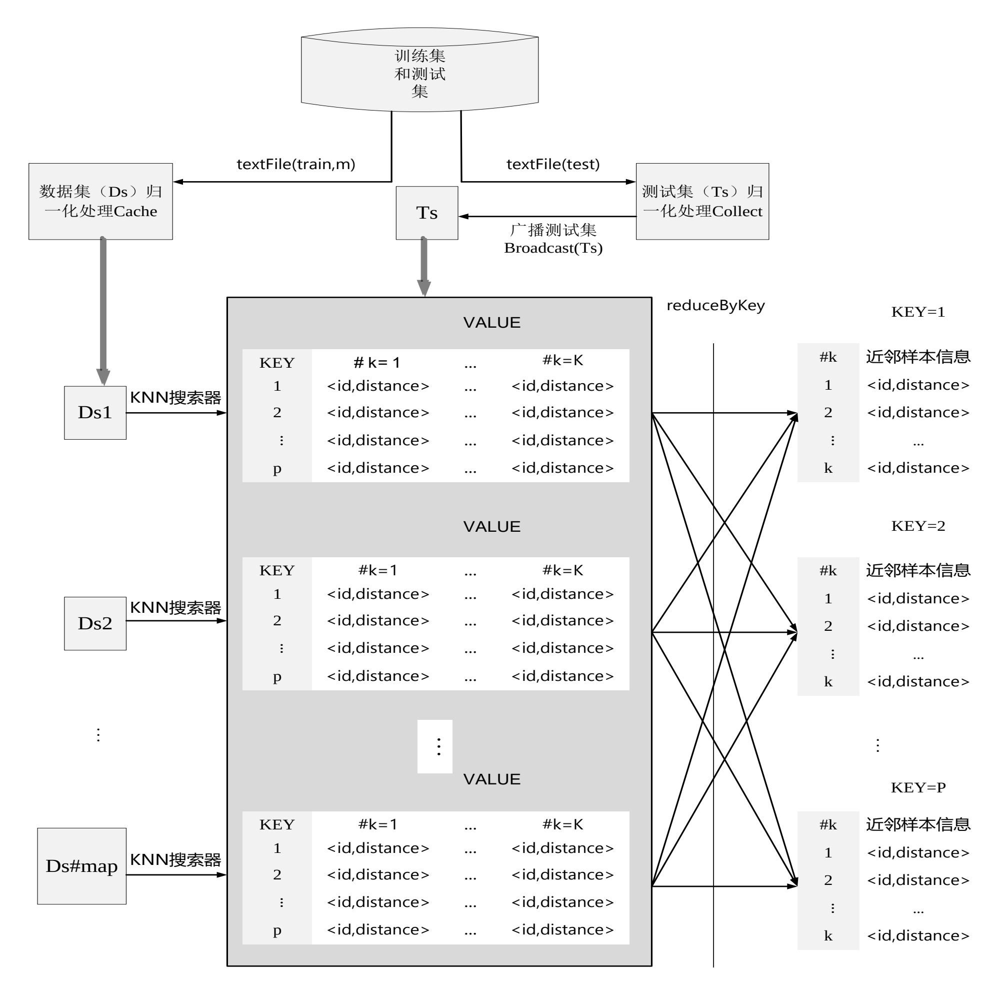


图 3　分布式 KNN 流程图

对于分类任务（也适用于回归），现有的方法比 KNN 连接方法更简单，因为它们不需要自己提供邻居，而只需要它们的类。到目前为止，已经提出了两种主要方法，他们都侧重于使用 Map 阶段将训练数据拆分为 m 个不相交的部分。例如，文献[14]提出了对每个单独的测试示例迭代重复 MapReduce 过程（没有明确定义的 reduce 函数），然而这在 Hadoop 和 Spark 中都非常耗时。文献[15]提出了一种称为 MR-KNN 的分类器，将其中一个 MapReduce 过程用来管理测试集分类。Hadoop 用于在映射阶段逐行读取测试数据。因此，这种模式具有可扩展性，但其性能可以通过内存中的解决方案进一步提高。

3　提出的方法

本文使用 Spark 给出一种用于大数据分类的替代分布式 KNN 模型，并将这种方法表示为基于 Spark 框架的大数据并行多标签 K 最近邻分类器设计。当训练集和测试集都是大数据集时，本文将重点放在减少 KNN 分类器的运行时间上。在并行

框架内计算 KNN 时，许多其他因素可能会影响执行时间，例如MapReduce 作业的数量 j 或需要的Map 和Reduce 的任务(m 和 r)数量。因此，在 Spark 中编写高效精确的 KNN 具有挑战性，并且必须考虑多个关键点才能获得高效且可扩展的模型。

本节介绍了 MapReduce 过程，该过程将管理适合内存的测试数据子集的分类。因此，这个 MapReduce 过程基于 MR-KNN，区别在于它允许多个 Reducer，检查运行所需的迭代以避免内存交换，并且在 Spark 下实现。作为 MapReduce 模型，这将计算划分为两个主要阶段：Map 和 Reduce。Map 阶段分割训练数据并针对每个块计算每个测试样本的 k 个最近邻居的距离和相应类别。Reduce 阶段将每个映射中 k 个最近邻居的距离聚合在一起，并给出 k 个最近邻居的确定列表。最终，执行 kNN 算法中的多数表决程序来预测结果类别。

3.1 Map 阶段

首先假设训练集 Ds 和相应的测试样本子集 Ts 先前已经从 HDFS 中作为 RDD 对象读取。因此，当训练数据集 Ds 被读取时，已经被分割成用户定义的 m 个不相交的子集。每个 map 任务 $(Map_1, Map_2, ..., Map_m)$ 利用训练集划分的每个块的样本来处理子集 Ds_i，其中 $1 \le i \le m$。因此，每个映射近似处理相似数量的训练实例。

为了获得 KNN 的精确实施，输入测试集 TS_i 不与训练集一起分割，这意味着 Ds_i 和 TS_i 都应该完全适合内存。在本文所述的 Spark 实现中，在每个分区用 $MapParttion$ 初始化一个 KNN 搜索器。与此同时，每个分区将获取整个测试集样本的局部 K 近邻。

每个映射都会发送多个输出，并允许使用多个 reducer。每个分区的结果数据以键值对（ $key, value$ ）形式存在。key 表示每个测样本的编号，而 value 表示以键值对（ $id, distance$ ）为元素的近邻数组。当使用的训练和测试数据集非常大时，使用更多 Reducer 可能会很有用。算法 1 描述了 Map 阶段的主要过程。其中 $file_{Ds}$ 表示训练集文件，$file_{Ts}$ 表示测试集文件，$RDD[id, List[(id, distance)]]$ 表示样本近邻集合。

算法 1　分布式 KNN Map 阶段过程

输入：$file_{Ds}$，$file_{Ts}$

输出：$RDD[id, List[(id, distance)]]$

$Ds \leftarrow textFile(file_{Ds}, \#map).map(normalize).map(LinearNNSearch).cache;$

$Ts \leftarrow textFile(file_{Ts}).map(normalize).map(key, value).collect;$

$arr \leftarrow sc.broadcast(Ts);$

$ans \leftarrow computeKNN(arr.Ds);$

3.2 Reduce 阶段

Reduce 阶段包括从 map 阶段提供的 TS_i 中的 k 个最近邻居。在 map 阶段结束后，对于测试样本 t 有 $\#map * K$ 个近邻，且所有具有相同键的元素都被分组。这个函数会一个接一个地处理这个列表中的每个元素，并用 k 个邻居更新结果列表 $results_{reducer}$。

由于来自 map 的矢量根据距离排序，所以更新过程变得更快。这包括合并两个排序列表以获得 k 的值，因此，最坏情况下的复杂度为 $O(k)$。该函数逐一比较每个邻居的每个距离值，从最近的邻居开始。如果距离小于当前值，则使用相应的值更新该位置的类别和距离，否则继续执行下面的值。算法 2 提供了 Reduce 操作的细节。

算法 2　分布式 Reduce 阶段过程

输入：$ans1[(id, distance)], ans2[(id, distance)];$

输出：$RDD_{res}[(id, Arrary[id])]$。

$res = newArray[type(id)][k+1];$

$i1 \leftarrow 0; i2 \leftarrow 0;$

for　$j \in \{0,1,2,...,k\}$

　if $(ans1(i1), distance \le ans2(i2).distance)$

　　$\{res(j) = ans1(i1).id; i1+=1;\}$

　else $\{res(j) = ans2(i2).id; i2+=1;\}$

总之，对于测试集中的每个实例，Reduce 函数都会根据之前描述的函数来聚合这些值。为了简化这个过程，我们使用 Spark 的 ReduceByKey（func）转换。

4　实验

4.1　实验配置

在这个实验研究中，我们将使用四个大数据分类数据集，PokerHand[16]、NUS-WIDE-bow[17]、Susy[18]和 Higgs[19]。因此，我们随机抽样所述数据集以获得各类别样本的平衡。这些数据集除了包含大量实例之外，它具有相对较多的特征，因此可以看到这个事实如何影响所提出的模型。

表 1 总结了这些数据集的属性，如样本数量、特征数量和标签数量信息。

表 1　数据集描述

数据集	样本数量	特征数量	标签数量
Susy	5,000,000	18	32
Higgs	11,000,000	28	42
PokerHand	1,025,010	10	60
NUS-WIDE-bow	269,648	500	81

对于实验研究，所有数据集均使用 5 折交叉验证（5-fcv）方案进行分区。这意味着数据集被分成 5 个子集，每子集中 80%的作为训练样本，其余的为测试实例。对于每个子集，kNN 算

法计算 TS 的最近邻居。在呈现的 MapReduce 方案中，数据集的实例数量和使用的 Map。数量有直接关系，因此 Map 数量越多，其中的实例数量越少。

测试实验都在由 16 个节点组成的集群上执行：主节点和 16 个计算节点。计算节点采用 2x Intel Xeon CPU E5-2620 处理器，所使用的软件及其配置的具体细节如下:

a) MapReduce implementations: Hadoop 2.6.0-cdh5.4.2 and Spark 1.5.1

b) Maximum number of map tasks: 256

c) Maximum number of reduce tasks: 128

d) Maximum memory per task: 2GB.

e) Operating System: Cent OS 6.5

请注意，可用内核的总数是 192，通过使用超线程技术将变为 384。因此，当算法探索大于 384 的映射时，不能期望线性加速，因为会有排队的任务。对于这些情况，本文将重点分析 Map 和 Reduce。

4.2 性能度量

本文用以下三种性能指标评估提出方法性能和可扩展性：

a)汉明损失。用以表示误分标签平均个数。算法性能越好，其汉明损失就越小。

$$hloss(h)=\frac{1}{p}\sum_{i=1}^{p}\frac{|h(x_i),\Delta Y_i|}{q} \tag{1}$$

其中：Δ 是两个集合之间的对称差分，q 是标签向量的长度，h 表示多标签分类器。

b）运行时间。根据训练数据集对给定的测试集进行分类，并记录 kNN 分类器花费的总时间。并行方法的总运行时间包括读取和分配所有数据，以及计算个最近邻居和多数投票。

c)加速比。为了证明并行算法与算法的顺序执行版本的效率，测量了顺序版本和并行版本的运行时间。根据 Amdahl 定律[20]，在完全平行的环境中，理论加速的最大值与使用的核心数量相同。

$$Speedup=\frac{base_line}{parallel_time} \tag{2}$$

其中：$base_line$ 是顺序版本所花费的总运行时间，而 $parallel_time$ 是使用并行版本后所花费的总运行时间。

F_{micro}^{β} **(基于标签的微观 F 值)**表示基于标签的精确度。

$$F_{micro}^{\beta}=F^{\beta}\left(\sum_{j=1}^{q}TP_j,\sum_{j=1}^{q}FP_j,\sum_{j=1}^{q}TN_j,\sum_{j=1}^{q}FN_j\right) \tag{3}$$

其中：TP、FP、TN、FN 是混淆矩阵中真实的正例、误判的正例、真实的反例，误判的反例。

4.3 与顺序版本 KNN 方法的比较

本节将本文方法与顺序版本 KNN 方法进行比较，其是基于 Hadoop MapReduce 框架。为此，本文使用 NUS-WIDE-bow 和 Higgs 数据集。NUS-WIDE-bow 和 Higgs 数据集是计算机视觉领域常用的数据集。在这些数据集中，本文方法只需要进行一次迭代，因为测试数据集适合每个映射的内存，并且 reducer 的数量也被固定为 1。

首先，本文在这些数据集上运行 kNN 的顺序版本作为基准。这个顺序版本逐行读取测试集，作为避免存储器问题的直接解决方案。表 2 显示了由标准顺序 KNN 算法随着邻居数量的不同所获得的运行时间（以秒为单位）和平均精度的结果（其中 Map 数量设置为 32）。

表 2　顺序 kNN 的性能表现

数据集	邻居数量	运行时间	平均精度
NUS-WIDE-bow	1	7989.34	0.67
	3	7972.93	0.62
	5	8076.26	0.64
	7	8023.27	0.76
Higgs	1	82627.36	0.73
	3	89023.34	0.74
	5	82334.92	0.71
	7	83261.35	0.72

表 3 总结了 $k=1$ 时的两种方法获得的结果（其中邻居数量设置为 1）。它显示了不同 Map 数量的平均总时间和相对于顺序版本实现的加速。如前所述，这两种方法都对应于 KNN 的精确实现。

表 3　顺序 KNN 和本文方法所得到的结果

数据集	Map 数量	顺序 KNN 平均运行时间（s）	本文方法 平均运行时间（s）	节省提升（s）
NUS-WIDE-bow	128	1978.26	311.24	1667.02
	64	3998.23	467.35	3530.88
	32	7989.34	992.42	6996.92
Higgs	256	11365.96	1509.43	9856.53
	128	22434.52	3163.92	19270.60
	64	43434.44	6132.61	37301.83

根据实验结果，做出以下分析：

正如在表 2 中可以看到的那样，在这两个数据集中，顺序版本的 KNN 方法所需的运行时间相当高。然而，表 3 显示了随着 Map 数量的增加，这两种方法的运行时间减少。与顺序版本 KNN 相比，本文方法的并行 KNN 的线速度提升速度更快。这是因为使用了内存中的数据结构，这使得我们可以避免从 HDFS 逐行读取测试数据。

比较顺序版本 KNN 和本文并行 KNN，结果显示本文基于

Spark 的并行 KNN 比基于 Hadoop 框架的顺序版本 KNN 运行时间减少了 6-8 倍。

4.4 与其他方法的性能比较

在基于 MapReduce 的现有分布式 kNN 模型中，将本文方法与 MR-KNN 方法进行比较，其是基于 Hadoop MapReduce 框架。MR-KNN 方法最初是为其他目的而设计的，而不是分类。他们还需要增加数据大小，甚至需要平方数量的 Map 任务。因此，该方案理论复杂度远高于所提出的技术。

表 4 给出了本文方法性能评估结果，表 5 给出了 MR-KNN 方法性能评估结果，其中 Map 与邻居数量分别设置为 128 和 1。从中可以看出，本文方法取得了较低的汉明损失，较少的运行时间，而且在加速比和 F^{β}_{micro} 两个指标数据也同样优于对比方法。证明了本文方法的有效性。

表 4　本文方法性能评估结果

数据集	汉明损失	运行时间/s	加速比	F^{β}_{micro}
Susy	0.025	4163.52	0.28	0.434
Higgs	0.022	3163.92	0.34	0.583
PokerHand	0.018	1134.23	0.31	0.463
NUS-WIDE-bow	0.021	311.24	0.26	0.341

表 5　MR-KNN 方法性能评估结果

数据集	汉明损失	运行时间/s	加速比	F^{β}_{micro}
Susy	0.034	6652.13	0.25	0.367
Higgs	0.032	5298.26	0.32	0.522
PokerHand	0.019	2214.22	0.30	0.387
NUS-WIDE-bow	0.024	638.23	0.24	0.323

5　结束语

K 最近邻分类器是数据挖掘中一种简单而有效的著名方法。由于时间和内存的限制，此模型在大数据领域的实际应用并不可行。在这项工作中，本文提供了基于 Apache Spark 框架的大数据并行多标签 K 最近邻分类器设计方法。映射阶段计算不同训练数据分割中的 *k* 个最近邻居，进一步在 reduce 阶段根据 map 阶段的结果确定最终的 K 近邻。最后并行地对近邻的标签集合进行聚合，通过最大化后验概率输出待预测样本的目标标签集合。在四个大数据分类数据集的测试结果也验证了提出方法的有效性。

参考文献：

[1] 董洋溢，李伟华，于会. 基于混合余弦相似度的中文文本层次关系挖掘 [J]. 计算机应用研究, 2017, 34 (5): 1406-1409. (Dong Yangyi, Li Weihua, Yu Hui. Hierarchical relation mining of Chinese text based on mixed cosine similarity [J]. Application Research of Computers, 2017, 34 (5): 1406-1409.)

[2] 党红恩，赵尔平，刘炜，等. 利用数据变换与并行运算的闭频繁项集挖掘方法 [J]. 湘潭大学自然科学学报, 2018, 40 (1): 119~122. (Dang Hongen, Zhao Erping, Liu Wei, *et al.* Closed frequent itemset mining method using data transformation and parallel operations [J]. Natural Science Journal of Xiangtan University, 2018, 40 (1): 119~122.)

[3] Seydell-Greenwald A, Raven E P, Leaver A M, *et al.* Diffusion imaging of auditory and auditory-limbic connectivity in tinnitus: preliminary evidence and methodological challenges [J]. Neural Plasticity, 2014, 2014 (2): 145943.

[4] 原继东，王志海，孙艳歌，等. 面向复杂时间序列的 k 近邻分类器 [J]. 软件学报, 2017, 28 (11): 3002-3017. (Yuan Jidong, Wang Zhihai, Sun Yangei, *et al.* K-Nearest Neighbor Classifier for Complex Time Series [J]. Journal of Software, 2017, 28 (11): 3002-3017.)

[5] Feng Huan, Eyers D, Mills S, *et al.* PCAF: scalable, high precision KNN search using principal component analysis based filtering [C]// Proc of International Conference on Parallel Processing. 2016: 638-647.

[6] Manjusha M, Harikumar R. Performance analysis of KNN classifier and K-means clustering for robust classification of epilepsy from EEG signals [C]// Proc of International Conference on Wireless Communications, Signal Processing and Networking. 2016: 2412-2416.

[7] Tiwari S, Kaushik S. Boundary points detection using adjacent grid block selection (AGBS) KNN-join method [C]// Proc of International Conference on Machine Learning and Data Mining. 2012.

[8] Demott P J, Prenni A J, Mcmeeking G R, *et al.* Integrating laboratory and field data to quantify the immersion freezing ice nucleation activity of mineral dust particles [J]. Atmospheric Chemistry & Physics, 2015, 14 (11): 393-409.

[9] Kumar A, Kiran M, Prathap B R. Verification and validation of MapReduce program model for parallel K-means algorithm on Hadoop cluster [C]// Proc of the 4th International Conference on Computing, Communications and Networking Technologies. IEEE, 2014: 1-8.

[10] 王淑艳，杨鑫，李克秋. MapReduce 框架下基于超平面投影划分的 Skyline 计算 [J]. 计算机研究与发展, 2014, 51 (12): 2702-2710. (Wang Shuyan, Yang Xin, Li Keqiu. Skyline Computing on MapReduce with Hyperplane-Projections-Based Partition [J]. Journal of Computer Research and Development, 2014, 51 (12): 2702-2710.)

[11] 陈珍，夏靖波，杨娟，等. 基于 MapReduce 的支持向量机态势评估算法 [J]. 计算机应用, 2016, 36 (1): 133-137. (Chen Zhen, Xia Jingbo, Yang Juan, *et al.* MapReduce-based Situation Assessment Algorithm for Support Vector Machines [J]. Journal of Computer Applications, 2016, 36 (1): 133-137.)

[12] Backman N, Pattabiraman K, Fonseca R, *et al.* C-MR: continuously executing MapReduce workflows on multi-core processors [C]// Proc of International Workshop on Mapreduce and Its Applications. 2012: 1-8.

[13] Zaharia M, Xin R S, Wendell P, *et al.* Apache Spark: a unified engine for big data processing [J]. Communications of the ACM, 2016, 59 (11): 56-65.

[14] Rogala M, Hidders J, Sroka J. DatalogRA: datalog with recursive aggregation in the spark RDD model [C]// Proc of International Workshop

on Graph Data Management Experiences and Systems. New York: ACM Press, 2016: 3.

[15] 卢选民，院文乐，邱杨，等. 一种改进的基于 KNN 的动态预测指纹定位算法 [J]. 计算机应用研究，2017，34 (7): 2016-2018. (Lu Xuanmin, Yuan Wenle, Qiu Yang, *et al*. An Improved dynamic prediction fingerprint location algorithm based on KNN [J]. Application Research of Computers, 2017, 34 (7): 2016-2018.)

[16] Jabin S. Poker hand classification [C]// Proc of International Conference on Computing, Communication and Automation. 2017: 269-273.

[17] Gu Yun, Xue Haoyang, Yang Jie, *et al*. Automatic Image Annotation Exploiting Textual and Visual Saliency [C]// Proc of International Conference on Neural Information Processing. Springer International Publishing, 2014: 95-102.

[18] Polpaya I C, Rao C L, Varughese S. Electromechanical behavior and microstructure of highly sensitive polyaniline//ethylene vinyl acetate composite Piezo-Resistive materials [C]// Proc of Conference on Smart Materials, Adaptive Structures and Intelligent Systems. 2016.

[19] Aad G, Al E, Bentvelsen S, *et al*. Observation of a new particle in the search for the standard model Higgs boson with the ATLAS detector at the LHC [J]. Physics Letters B, 2012, 716 (1): 1-29.

[20] Das A K, Jaeki H, Goswami S, *et al*. Augmenting Amdahl's Second Law: A Theoretical Model to Build Cost-Effective Balanced HPC Infrastructure for Data-Driven Science [C]// Proc of IEEE International Conference on Cloud Computing. 2017: 147-154.